Main.cpp

```
#include "Game.h"

int main(int, char**)
{
    // The magic number is meant to prevent Game::main() from being
    // called by the subclassed game program.
    GE161::Game::theGame->main(8675309);
    return 0;
}
```

Game.cpp

```
//The linker wants this static variable defined here; it will get
// its actual value in main().
GE161::Game* GE161::Game::theGame = nullptr;

GE161::Game::Game()
{
    // Store a pointer to the game in a static variable for access by
main().
    if (theGame == nullptr)
    {
        theGame = this;
    }
    else
    {
        fatalSDLError("More than one game object has been created.");
    }
}
```

Game-main.cpp

```
#include "Game.h"

void GE161::Game::main(int x)
{
    if (x != 8675309)
    {
        fatalSDLError("Do not invoke GE161::Game::main()");
        return;
    }

    if (SDL_Init(SDL_INIT_EVERYTHING) != 0)
    {
        fatalSDLError("SDL_Init(SDL_INIT_EVERYTHING) Error in main(): ",
SDL_GetError());
        return;
    }
```

```
    // Invoke the game's overridden setup() method, and create the SDL
window.
    window_ = new GE161::Window();
    setup();
    window_->initialize();
    window_->clearBackground();

    eventQueue_ = new GE161::EventQueue();
```

```
std::string sceneName = GE161::Game::START_GAME;
int returnCode = CONTINUE_SCENE;

// Loop through the outer while loop once per scene.
// A negative return code from a scene says "Stop the program!"
while (returnCode >= 0)
{
    // Ask the game which scene to do next.
    sceneName = GE161::Game::theGame->chooseScene(
                                            sceneName, returnCode);
    GE161::Scene* scene =
                GE161::Game::theGame->lookUpScene(sceneName);

    // Now we have the scene to use.  First, run its setup().
    bool success = scene->setup();
    if (!success)
    {
        fatalSDLError("error in main(), setup failed for scene",
                            sceneName);
        return;
    }
```

```
        // setup() succeeded.  Run its draw() until a non-0 return code.
        returnCode = CONTINUE_SCENE;

        // Loop through the inner while loop once per frame in scene.
        while (returnCode == CONTINUE_SCENE)
        {
            eventQueue_->getSDLEvents();
            eventQueue_->callEventListeners();

            window_->clearBackground();
            returnCode = scene->draw();
            window_->drawToScreen();
        }
        std::string m = std::string("return code from ") + sceneName +
                        std::string(": ") + std::to_string(returnCode);
        debugOut(m);
    }

    SDL_Quit();
    return;
}
```

Sprite.h

```
namespace GE161
{
    class Sprite
    {
        friend class GameObject;

    public:
        Sprite(int frameWidth, int frameHeight);
        ~Sprite();

        int makeFrame(std::string fileName, int x, int y);
        int addFrameToSequence(std::string sequenceName, int frameIndex);
    private:

        // The sequenceList map is use to map a sequenceName string to
        // a vector of frameIndexes, and the index of the current fIndex
        std::map<std::string, std::pair<std::vector<int>, unsigned int>>
                                    sequenceList;
        std::map<std::string, SDL_Texture*> storedTextures;
    };
}
```

GameObject.cpp

```
#include "GameObject.h"
#include "Game.h"

GE161::GameObject::GameObject(int startingX, int startingY) :
x_(startingX),
y_(startingY),
sprite_(nullptr)
{
}


GE161::GameObject::~GameObject()
{
}

void GE161::GameObject::setPos(int x, int y)
{
    x_ = x;
    y_ = y;
}
```

```
void GE161::GameObject::attachSprite(GE161::Sprite* sprite)
{
    sprite_ = sprite;
}

void GE161::GameObject::draw(int frameIndex)
{
    if (sprite_ == nullptr)
    {
        debugOut("GameObject::draw called, but no Sprite is attached.");
        return;
    }

    SDL_Rect srcrect = {sprite_->frames[frameIndex].x,
                        sprite_->frames[frameIndex].y,
                        sprite_->frameWidth_, sprite_->frameHeight_};
    SDL_Rect destrect = {x_, y_, ……};
    int success = SDL_RenderCopy(GE161::Game::theGame->getRenderer(),
              sprite_->frames[frameIndex].texture, &srcrect, &destrect);
    if (success != 0)
    {
     fatalSDLError("In GameObject::draw, SDL_RenderCopy: ", SDL_GetError());
    }
}
```

```
void GE161::GameObject::draw(std::string sequenceName)
{
    draw(sprite_->getNextFrameIndex(sequenceName));
}
```

EventListener.h

```
    class EventListener
    {
    public:
        EventListener();

        virtual void onEvent(Event& ev);
    };
```

Scene.h

```
namespace GE161
{
    class Scene : public EventListener
    {
    public:
        Scene();
        ~Scene();

        virtual bool setup() = 0;
        virtual int draw() = 0;
    };

}
```

EventQueue.cpp

```
void GE161::EventQueue::getSDLEvents()
{
    SDL_Event e;
    while (SDL_PollEvent(&e))
    {
        eventQueue_.push(new GE161::Event(e));
    }
}

void GE161::EventQueue::registerAsListener(int eventType, EventListener*
listener)
{
    std::pair<int, EventListener*> p(eventType, listener);
    eventListenerList_.push_back(p);
}
```

EventQueue.cpp

```
void GE161::EventQueue::callEventListeners()
{
    // for each event in the queue
    while (!eventQueue_.empty())
    {
        Event* e = eventQueue_.front();

        // for each listener
        for (auto p : eventListenerList_)
        {
            int listenedForEventType = p.first;
            EventListener* listener = p.second;
            // if there is a match between this event type and
            // the listener's event type
            if (listenedForEventType == e->type)
            {
                // call the listener's onEvent()
                listener->onEvent(*e);
            }
        }
        // and now we have handled that event
        eventQueue_.pop();
    }
}
```

Event.h

```
class Event
{
    friend class EventQueue;

public:
    Event();
    Event(SDL_Event& e);
    ~Event();

    // These fields are public for convenient access.
    int type;
    std::string key;
    int mouseButton;
    int mouseClicks;
    int x;
    int y;

    // event types
    static const int QUIT = SDL_QUIT;
    static const int KEY_DOWN = SDL_KEYDOWN;
    static const int KEY_UP = SDL_KEYUP;
    static const int MOUSE_MOTION = SDL_MOUSEMOTION;
    static const int MOUSE_BUTTONDOWN = SDL_MOUSEBUTTONDOWN;
```

```
        static const int MOUSE_BUTTONUP = SDL_MOUSEBUTTONUP;

    private:
        SDL_Event sdl_event;

    };

}
```

Event.cpp

```
GE161::Event::Event(SDL_Event& e) :
    sdl_event(e),
    key(""),
    mouseButton(0),
    mouseClicks(0),
    x(0),
    y(0)
{
    type = e.type;  // copy SDL type
    if (type == KEY_UP || type == KEY_DOWN)
    {
        switch (e.key.keysym.sym)
        {
        case SDLK_0: key = "0"; break;
        case SDLK_9: key = "9"; break;
        case SDLK_a: key = "A"; break;
        case SDLK_z: key = "Z"; break;
        case SDLK_SPACE: key = " "; break;
        case SDLK_DOWN: key = "DOWN"; break;
        case SDLK_LEFT: key = "LEFT"; break;
        case SDLK_RIGHT: key = "RIGHT"; break;
        case SDLK_UP: key = "UP"; break;
        // many more symbols need to be accounted for.
```

```
            // see https://wiki.libsdl.org/SDL_Keycode
            }
            return;
        }
        if (type == MOUSE_BUTTONUP || type == MOUSE_BUTTONDOWN)
        {
            mouseButton = e.button.button;
            mouseClicks = e.button.clicks;
            x = e.button.x;
            y = e.button.y;
            return;
        }
        if (type == MOUSE_MOTION)
        {
            x = e.motion.x;
            y = e.motion.y;
            return;
        }
}
```

GE161helpers.cpp

```cpp
#include "GE161int.h"

void fatalSDLError(std::string message)
{
    SDL_ShowSimpleMessageBox(SDL_MESSAGEBOX_ERROR,
        "SDL Error",
        message.c_str(),
        NULL);
}

void fatalSDLError(std::string message1, std::string message2)
{
    SDL_ShowSimpleMessageBox(SDL_MESSAGEBOX_ERROR,
        "SDL Error",
        (message1 + " " + message2).c_str(),
        NULL);
}

// This is Windows and Visual Studio dependent.
void debugOut(std::string message)
{
    OutputDebugString((message + "\n").c_str());
}
```